

COMPTE-RENDU TP N°3

I) PREPARATION A FAIRE AVANT LE TP

Utilisation de PYGAME

```
import pygame

pygame.init()

ecran = pygame.display.set_mode((300, 200))

pygame.quit()
```

(1) Lorsque l'on lance ce code, une fenêtre d'affichage apparaît et disparaît aussitôt.

La première ligne permet d'importer le module Pygame, la seconde ligne permet d'initialiser le module. La troisième permet d'initialiser une fenêtre, la quatrième ligne referme cette dernière.

```
import pygame

pygame.init()

ecran = pygame.display.set_mode((300, 200))

continuer = True

while continuer:

    for event in pygame.event.get():

        if event.type == pygame.KEYDOWN:

            continuer = False

pygame.quit()
```

- (2) Lorsqu'on lance ce code, une fenêtre d'affichage apparaît. Elle disparaît lorsque l'on appuie sur une touche du clavier. En effet, la boucle while empêche la fenêtre de se fermer tant que la condition `if event.type == pygame.KEYDOWN` n'est pas vérifiée

Utilisation de PYOPENGL

```
import pygame

import OpenGL.GL as gl

import OpenGL.GLU as glu

if __name__ == '__main__':

    pygame.init()

    display=(600,600)

    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)

    # Sets the screen color (white)

    gl.glClearColor(1, 1, 1, 1)

    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces

    gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)

    gl.glEnable(gl.GL_DEPTH_TEST)

    # Placer ici l'utilisation de gluPerspective.

    glu.gluPerspective(45,display[0]/display[1],0.1,50.0)

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                exit()
```

(1) Pas de message d'erreur avec le code ci-dessus

```
gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode
lignes (segments)

gl.glColor3fv([0, 0, 0]) # Indique la couleur du prochain segment en RGB

gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne

gl.glVertex3fv((1, 1, -2)) # Deuxième vertice : fin de la ligne

gl.glEnd() # Find du tracé

pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
```

(2) En incorporant ce code, pygame affiche une fenêtre contenant un segment noir (couleur [0, 0, 0]) en diagonale de la fenêtre.

Donc pour obtenir les 3 axes x, y et z respectivement en rouge, vert et bleu, on rentre le code suivant :

```
import pygame

import OpenGL.GL as gl

import OpenGL.GLU as glu


if __name__ == '__main__':

    pygame.init()

    display=(600,600)

    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)

    # Sets the screen color (white)

    gl.glClearColor(1, 1, 1, 1)

    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
```

```
gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)

gl.glEnable(gl.GL_DEPTH_TEST)

glu.gluPerspective(45,display[0]/display[1],0.1,50.0)

gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en
mode lignes (segments)

gl.glColor3fv([255, 0, 0]) # Couleur rouge de l'axe x

gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de l'axe (origine)

gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : direction de l'axe

gl.glColor3fv([0, 255, 0]) # Couleur verte de l'axe y

gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de l'axe (origine)

gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : direction de l'axe

gl.glColor3fv([0, 0, 255]) # Couleur rouge de l'axe x

gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de l'axe (origine)

gl.glVertex3fv((0, 0, -3)) # Deuxième vertice : direction de l'axe

gl.glEnd() # Find du tracé

pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique

while True:


    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            exit()
```

On remarque que l'axe z est défini mais n'apparaît pas sur la représentation 2D de Pygame, ce qui est normal car ce dernier est en perspective

 pygame window

— □ ×




- (3) En utilisant la fonction `glTranslatef`, on constate que nos axes sont déplacés selon les coordonnées du vecteur que l'on rentre en argument.

Avant d'effectuer la rotation, on effectue une translation pour que la rotation autour des vecteurs des axes soit « centrée ». On peut ainsi faire apparaître l'axe Z avec une rotation de 90° degrés autour de l'axe x avec le code suivant implémenté juste après la fonction `gluPerspective` :

```
gl.glTranslatef(0.0, 2, -5)

gl.glRotatef(-90, 1, 0, 0)
```

 pygame window

— □ ×



Suite à la rotation de 90° sur l'axe x, on observe bien l'axe z

ENVIRONNEMENT DE TRAVAIL DU TP

Q1a) : le code fonctionne, les touches a, z et Z permettent respectivement de faire apparaître ou disparaître les axes, et faire une rotation (positive pour z et négative pour Z) de $2,5^\circ$ autour de l'axe z

Q1b) :

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display()
```

Ce code correspond à l'affichage (.display()) des axes selon une configuration de couleur et de position de la fenêtre.



Les différents setters « setParameter() » vont imposer d'autres configurations pour l'affichage

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
    setParameter('xAxisColor', [1, 1, 0]). \
    setParameter('yAxisColor', [0,1,1]). \
    display()
```



Le chaînage des méthodes `setParameter()` et `Display()` est possible car ces méthodes prennent en argument l'objet `self`, et retournent également ce dernier.

Traitement particulier pour `screenPosition` car par défaut, c'est un paramètre réglé à -10, ce qui donne une image lointaine. C'est donc pour augmenter la visibilité.

Q1c) Il faut ajouter l'instruction :

`gl.glRotatef(90, 1.0, 0.0, 0.0)`

à la méthode `initializeTransformationmatrix`. De cette manière, l'axe x sera représenté horizontalement, l'axe y devient alors la profondeur.

II) MISE EN PLACE DES INTERACTIONS AVEC L'UTILISATEUR DE PYGAME

Q1d) Gestion des touches « page up » et « page down » :

On définira respectivement les touches u et d pour zoomer ou dézoomer

La documentation nous renseigne sur la fonction `glScalef` qu'on utilise dans le code suivant :

```
If self.event['unicode'] == u :  
  
    gl.glScalef(1.1,1.1,1.1)  
  
Elif self.event['unicode'] == d :  
  
    gl.glScalef(1/1.1,1/1.1,1/1.1)
```

Ainsi, on zoom en appuyant sur la touche 'u' et on dézoom en appuyant sur la touche 'd'

Q1e) Afin de pouvoir zoomer ou dézoomer à l'aide de la molette de la souris, on procède par analogie avec la question précédente pour mettre au point le code suivant :

```
Def processMouseButtonDownEvent(self) :  
  
    If self.event.button == 4 :  
  
        gl.glScalef(1.1,1.1,1.1)  
  
    Elif self.event.button == 5 :  
  
        gl.glScalef(1/1.1,1/1.1,1/1.1)
```

Ainsi, lorsqu'on utilise la molette, il est possible de zoomer et dézoomer l'affichage de la fenêtre pygame

Q1f)

L'objectif est de réaliser une translation selon l'axe x composé à l'axe z lorsque le bouton droit est enfoncé, et de réaliser une rotation selon l'axe x composé à l'axe z lorsque le bouton gauche de la souris est enfoncé

Pour ce faire, le code ci-dessous fonctionne :

```
Def processMouseMotionEvent(self) :  
  
    If pygame.mouse.get_pressed()[2] == 1 :  
  
        gl.glTranslatef(self.event.rel[0]/20, 0, 0)  
  
        gl.glTranslatef( 0, 0,-self.event.rel[1]/20)  
  
    Elif pygame.mouse.get_pressed()[0] == 1 :  
  
        gl.glRotate(self.event.rel[0], 0, 0, 1)  
  
        gl.glRotate(self.event.rel[1], 1, 0, 0)
```

Le bouton droit de la souris permet alors de déplacer le repère, le clic gauche permet de réaliser une rotation du repère sans le déplacer.

III) CREATION D'UNE SECTION

Q2a) On écrit le code correspondant à la méthode generate permettant de créer les sommets et faces de la section :

```
# Defines the vertices and faces
def generate(self):
    self.vertices = [
        [0, 0, 0 ], #0#
        [0, 0, self.parameters['height']], #1#
        [self.parameters['width'], 0, self.parameters['height']], #2#
        [self.parameters['width'], 0, 0], #3#
        [0, self.parameters['thickness'], 0 ], #4#
        [0, self.parameters['thickness'], self.parameters['height']], #5#
        [self.parameters['width'], self.parameters['thickness'], 0 ], #6#
        [self.parameters['width'], self.parameters['thickness'], self.parameters['height']] #7#
    ]
    self.faces = [
        [0, 3, 2, 1],
        [0, 1, 5, 4],
        [0, 4, 6, 3],
        [3, 2, 7, 6],
        [1, 5, 7, 2],
        [4, 5, 7, 6]
    ]
```

Q2b)

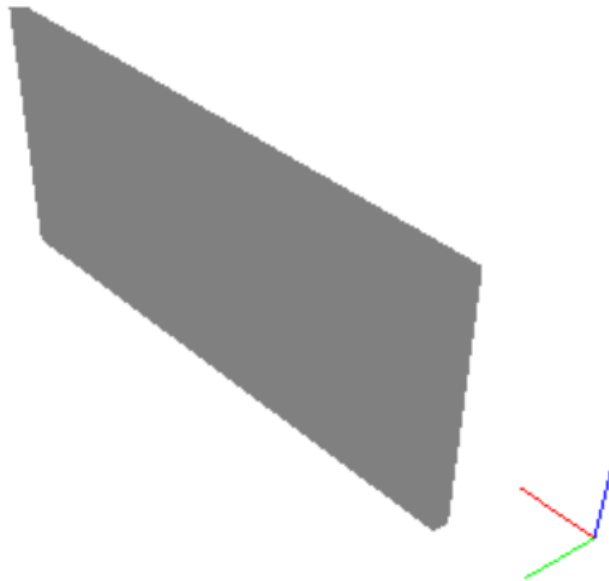
L'instruction **Configuration().add(section).display()** permet de créer et d'afficher une section à l'aide de la méthode draw de la classe section

Le code de la fonction draw de la classe section est le suivant :

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2])

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS)
    gl.glColor3fv([0.5, 0.5, 0.5])# Tracé d'un quadrilatère
    for f in self.faces:
        for x in f:
            gl.glVertex3fv(self.vertices[x])
    gl.glEnd()
    gl.glPopMatrix()
```

En lançant le script il apparait la section suivante :



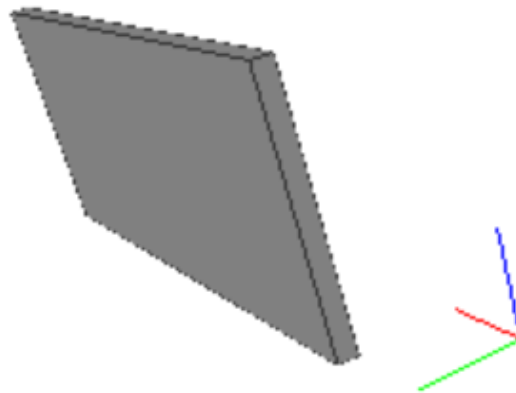
Q2c) Méthode drawEdges (arrêtes)

```
# Draws the edges
def drawEdges(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS)
    gl.glColor3fv([0.25, 0.25, 0.25]) # Tracé d'un quadrilatère
    for f in self.faces:
        for x in f:
            gl.glVertex3fv(self.vertices[x])
    gl.glEnd()
    gl.glPopMatrix()
```

On modifie la méthode draw() afin que la méthode drawEdges soit réalisée en 1er lorsque le paramètre **edges**, fourni au constructeur ou via le « setter » **setParameter()**, prend la valeur **True** :

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2])
    if self.parameters['edges'] == True:
        self.drawEdges()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS)
    gl.glColor3fv([0.5, 0.5, 0.5]) # Tracé d'un quadrilatère
    for f in self.faces:
        for x in f:
            gl.glVertex3fv(self.vertices[x])
    gl.glEnd()
    gl.glPopMatrix()
```

En lançant le script, on peut visualiser les arrêtes :



IV) CREATION DES MURS

Q3a) Analyse du fichier wall.py :

Le constructeur crée une liste « **objects** » contenant les différentes sections.

```
self.parameters['color'] = [0.5, 0.5, 0.5]

# Objects list
self.objects = []

# Adds a Section for this object
self.parentSection = Section({'width': self.parameters['width'], \
                                'height': self.parameters['height'], \
                                'thickness': self.parameters['thickness'], \
                                'color': self.parameters['color'], \
                                'position': self.parameters['position']})
self.objects.append(self.parentSection)

# Getter
```

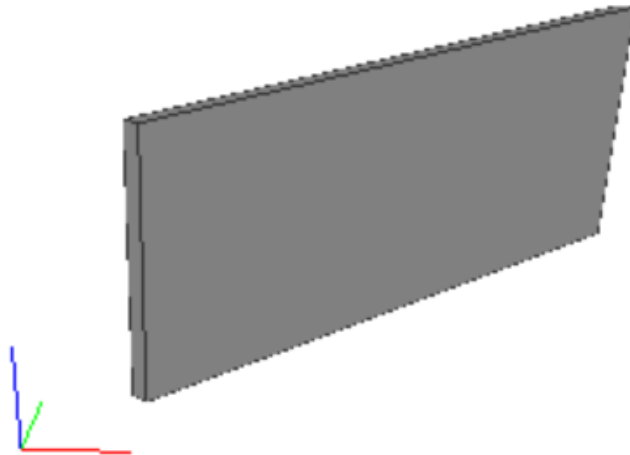
Ecriture de la fonction draw :

```
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    for x in self.objects:
        if (isinstance(x,Section)) == True:
            gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
            x.parameters['edges'] = True
            x.drawEdges
            x.draw()
        elif():
            x.parameters['edges'] = True
            x.drawEdges
            x.draw()
    gl.glPopMatrix()
```

Ecriture de la fonction Q3a) :

```
def Q3a():  
    return Configuration().add(Wall({'position': [1, 1, 0], 'width':7, 'height':2.6, 'orientation':45}))
```

En lançant le script, on retrouve bien notre mur avec une orientation modifiée de 45°



V) CREATION D'UNE MAISON

Code draw() de la classe house :

```
# Draws the house  
def draw(self):  
    # A compléter en remplaçant pass par votre code  
    gl.glPushMatrix()  
    for x in self.objects:  
        x.draw()  
    gl.glPopMatrix()
```

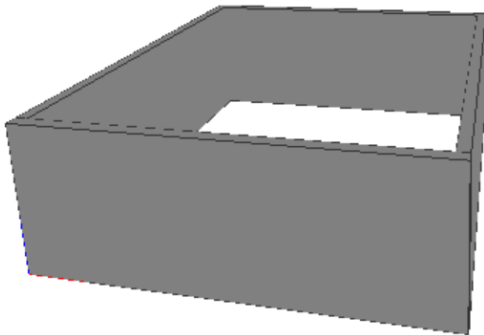
Q4a) on modifie le code initial de Q4a :

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width':7, 'height':2.6, 'orientation':0})
    wall2 = Wall({'position': [0, 5, 0], 'width':7, 'height':2.6, 'orientation':0})
    wall3 = Wall({'position': [0, 0, 0], 'width':5.20, 'height':2.6, 'orientation':90})
    wall4 = Wall({'position': [7, 0, 0], 'thickness':5.2, 'height':2.6, 'orientation':0})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

Remarque : j'ai modifié le constructeur de la classe house de sorte à pouvoir tracer les 4 murs d'une maison plus facilement :

```
# Sets the default parameters
if 'position' not in self.parameters:
    self.parameters['position'] = [0, 0, 0]
if 'width' not in self.parameters:
    self.parameters['width'] = 0.2
if 'height' not in self.parameters:
    raise Exception('Parameter "height" required.')
if 'orientation' not in self.parameters:
    self.parameters['orientation'] = 0
if 'thickness' not in self.parameters:
    self.parameters['thickness'] = 0.2
if 'color' not in self.parameters:
    self.parameters['color'] = [0.5, 0.5, 0.5]
```

Ainsi, lorsque on lance le script, on obtient le tracé suivant :



VI) CREATION D'OUVERTURES

Q5a) Nous allons procéder manière analogue avec la méthode utilisée pour réaliser les sections.

On commence par écrire la methode generate() de la classe opening :

```
def generate(self):
    self.vertices = [
        [0,0,0],
        [0,0, self.parameters['height']],
        [self.parameters['width'],0, self.parameters['height']],
        [self.parameters['width'],0,0],
        [0,self.parameters['thickness'],0],
        [0,self.parameters['thickness'], self.parameters['height']],
        [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
        [self.parameters['width'], self.parameters['thickness'],0]]

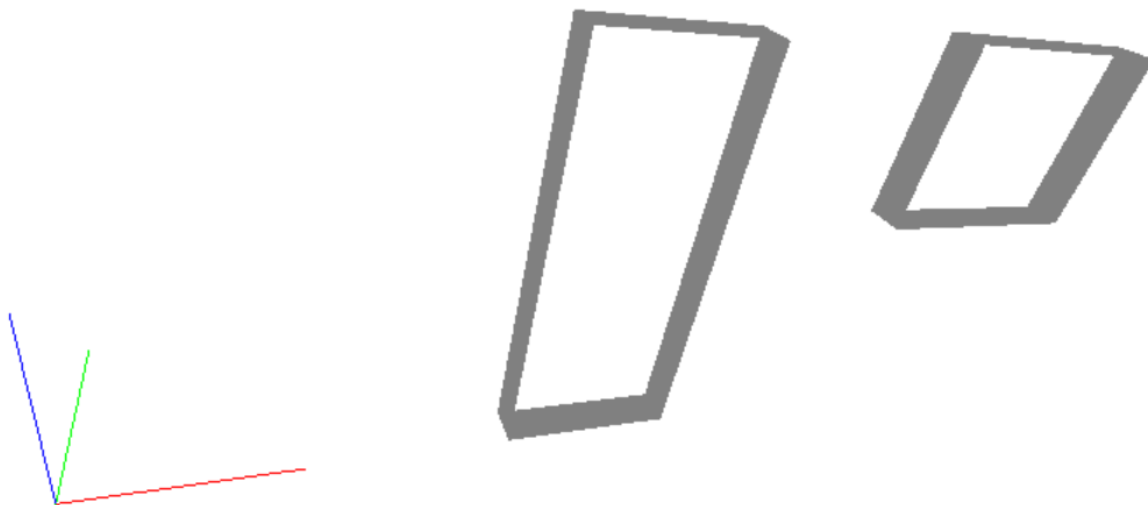
    self.faces = [[0,4,5,1],
                  [2,3,7,6],
                  [1,2,6,5],
                  [0,3,7,4]]
    ]
```

Puis la méthode draw() de la classe opening :

```
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) #couleur
    for f in self.faces:
        for x in f:
            gl.glVertex3fv(self.vertices[x])
    gl.glEnd()
    gl.glPopMatrix()
```

On lance le script qui nous donne alors le tracé suivant :



VII) Conclusion :

A travers ce travail, j'ai pu avoir un aperçu des méthodes utilisables en python pour la représentation d'objets 3D.

Je n'ai malheureusement pas pu traiter ce TP dans son intégralité, établir un lien entre les lignes de code et la visualisation dans l'espace est une tâche qui m'a coûté beaucoup de temps.